

Real-time Operating Systems and Systems Programming

Programming an Operating System Lecture 9

Summary

- Programming
 - Tasks
 - Scheduling
 - Context Switches
 - Mutexes
- Michael Barr, Programming Embedded Systems in C and C++

A Misconception

- **“Operating system internals are complex.”**
 - Software companies like the thought
 - Speed
- Embedded OS even more simple
 - Single user
 - No disk drivers & filesystem
 - No GUI

Os under discussion ADEOS

ftp://ftp.oreilly.com/examples/nutshell/embedded_c/

- A Decent Embedded Operating System

Tasks

- Tasks are parallel in only seemingly
- Book analogy
- For a task you have to keep its state somewhere
 - Pointer to next instruction
 - Address of the top of stack (stack pointer)
 - Contents of processors flag & general registers
- Task control block usually, but object in example

Task States

- Ready <> Running > Waiting > Ready
- Running-Ready – task switching
- Waiting = Blocking
- Only one running at time

Task Creation

- ADEOS allows only task creation
- When (and if) a task function returns, task is deleted
- Construction needs a function, priority and stack size.

Scheduler

- Decides which task gets to run
- Example uses priority list for task scheduling
- FIFO behaviour in case of conflicts
- 255 priority levels

Scheduling Points

- Events during which scheduler is invoked
- (Task creation has one)
- `os.schedule()` runs

Clock Tick

- Runs on timer interrupts
- Waking the tasks which wait for timer to expire

Ready List

- Ordinary linked list
- Priority queue
- Next task always on top

Idle task

- Empty loop
- Hidden from application developer
- Has id and priority of 0
- Always ready
-

schedule()

- Looks for top task, if it is the running task, good
- Otherwise switches tasks
- Note that tasks start only after initialization of scheduler (since scheduler is invoked on task creation too)

Example of task creation

Context Switch

- Architecture specific
- Must be written in assembler language
- `restoreContext()` and `saveContext()`
- Tasks wake in `saveContext()` and a clever jump is done to distinguish between saving and restoring.

Mutexes

- Multitasking aware binary flag
- Setting and clearing are atomic
- Interrupts are disabled
- Implementation: flag + waiting list
- Initialization simple

Mutex setting and clearing

- Setting
 - If taken, process goes to waiting state until released
 - Scheduling called
- Releasing
 - Does not block
 - On release a context switch might occur due to scheduling

Critical section

- Deadlock?
- Priority inversion?